ELSEVIER

Discrete Applied Mathematics 72 (1997) 115–139

DISCRETE APPLIED MATHEMATICS

Algorithms for minclique scheduling problems ☆

Bernd Jurisch [a], Wieslaw Kubiak [a,*], Joanna Józefowska [b]

[a] *Faculty of Business Administration, Memorial University of Newfoundland, Canada*
[b] *Institute of Computing Science, Technical University of Poznań, Poland*

Received 8 March 1994; revised 30 March 1995

Abstract

We consider a family of well-known scheduling problems that reduce to the problem of finding a minimum weighted clique in a complete weighted graph with negative weights and self-loops allowed. We present a uniform algorithmic approach to finding optimal as well as suboptimal solutions for these problems. Also, we report results of computational tests for suboptimal algorithms developed in the paper.

Keywords: Single/multiple machine scheduling; Minimum weighted clique; Dynamic programming; Spectral algorithms; Tabu search

1. Introduction

We consider four scheduling problems: scheduling two machines to minimize makespan (MAKS), scheduling two machines to minimize total weighted completion time (WCT), scheduling a single machine to minimize completion time variance (CTV), and scheduling a single machine to minimize total weighted earliness/tardiness (WET). All have been shown to be NP-hard in the ordinary sense [15, 17, 16, 8], and there have been pseudopolynomial time algorithms proposed [18, 12, 8] for each.

We exploit similarities between these problems in order to give a uniform algorithmic approach to finding their optimal as well as suboptimal solutions. In particular, we prove that each of these scheduling problems reduces to the problem of finding a minimum weighted clique in a complete weighted graph with negative weights and self-loops allowed (MinClique). The latter is closely related to the problem of finding a maximum cut in a complete weighted graph with non-negative weights (MaxCut) [1]. Actually, we show that MaxCut reduces to MinClique. We propose a family of

☆ This research has been supported by the Natural Sciences and Engineering Research Council of Canada under Grant OGP0105675. In addition, Dr. Józefowska has been supported by the State Committee for Scientific Research Poland Grant 8 T11F 010 08p02.

* Corresponding author. E-mail: wkurbiak@kean.ucs.mun.ca.

SSDI 0166-218X(96)00040-6

heuristics to solve the MinClique problem and, thus, all the scheduling problems at one fell swoop. These heuristics, we call them spectral algorithms, exploit the eigenvectors of the adjacency matrices of the graphs being outcomes of the reductions. Similar algorithms have been earlier developed for the MaxCut problem [1, 2, 22], see also [21] for an excellent review.

The idea behind our heuristics is to use eigenvectors to generate a set of solutions, with polynomially bounded cardinality, evaluate them, and choose the best one. This, however, may provide an outcome that is not a local optimum, thus, one more stage is usually required to provide better suboptimal solution. We present a simple job exchange procedure and a tabu search procedure for this stage.

Since all the problems under consideration are NP-hard it is time-consuming to obtain optimal solutions for them for instances with more than twenty jobs. We do not use the existing pseudopolynomial time optimization algorithms for the same reason; we believe that "difficult" instances of the problems are those with long jobs and heavy weights, thus, we generate them uniformly between 1 and $2^{n/k}$, where n is the number of jobs and $k \in \{1,2,4\}$ (see also [20], p. 128). This makes the pseudopolynomial time algorithms inefficient even for relatively small numbers of jobs. We have observed through our tests that instances with processing times and weights bounded by a linear function of n were very likely solved to optimality by our heuristics. We compared accuracy of solutions given by the spectral algorithms with the accuracy of solutions given by the well-known, and often used as benchmarks, heuristics: LPT and the differencing algorithm of Karmarkar and Karp [14] for MAKS, the Smith's rule [23] for WCT, the Kanet's [13] algorithm for CTV, and a modified Smith's algorithm for WET. Each of these algorithms has been appended by the same tabu search as the spectral algorithms for MinClique.

We exploit the similarities between the scheduling problems further; we have observed that each of the four reductions to MinClique results in a graph in which the weight associated with edge (i,j) is of the form $a_i * b_j$. This multiplicative form makes it possible to develop a pair of pseudopolynomial time algorithms for the resulting MinClique problem and, thus, also for each scheduling problem. One algorithm defines a state in terms of a_i's and weights them using b_j's, the other defines a state in terms of b_j's and weights them using a_i's. We shall call this pair a row-column (or (r-c)) pair. Such a pair for CTV is given in Kubiak (1991).

The paper is organized as follows. In Section 2, we formulate scheduling and graph-theoretic problems considered in this paper, and proof polynomial time reducibility between the problems. In Section 3, we present an (r-c) pair of dynamic programs for the special case of MinClique with weights in the multiplicative form signaled above. Section 4 is devoted to the presentation of spectral algorithms for MinClique; spectral algorithms for MaxCut are also discussed. In Section 5, we present a simple job exchange procedure and a tabu search algorithm for improving suboptimal solutions generated by the spectral algorithms. Section 6 presents computational results of tests of the spectral algorithms. The paper ends with concluding remarks in Section 7.

2. Reductions to MinClique

Let there be n jobs indexed $1,\ldots,n$ with processing times $p_1, p_2, \ldots, p_n$ and weights $w_1,\ldots,w_n$, respectively. The jobs are to be scheduled in such a way that a certain objective function of the completion times C_i of the jobs is minimized. We consider the following problems:

Scheduling two machines to minimize makespan (MAKS): Find a schedule of the jobs on two machines M_1 and M_2 such that the maximum completion time $\max_{1\leqslant i\leqslant n} C_i$ is minimized.

Scheduling two machines to minimize total weighted completion time (WCT): Find a schedule of the jobs on two machines M_1 and M_2 such that the weighted completion time $\sum_{i=1}^{n} w_i C_i$ is minimized.

Scheduling one machine to minimize completion time variance (CTV): Find a schedule of the jobs on one machine such that the completion time variance

$$\frac{1}{n}\sum_{i=1}^{n}\left(C_i - \frac{1}{n}\sum_{i=1}^{n} C_i\right)^2$$

is minimized.

Scheduling one machine to minimize total weighted earliness/tardiness (WET): Find a schedule of the jobs on one machine such that the weighted sum of earliness/tardiness $\sum_{i=1}^{n} w_i|C_i - d|$ is minimized, for a given common due date $d \geqslant \sum_{i=1}^{n} p_i$.

Throughout this paper, X denotes the set of all 0,1 vectors $(x_1,\ldots,x_n)$, i.e., $X = \{0,1\}^n$. We also consider the following graph-theoretic problems.

Maximum weighted cut in a complete graph (MaxCut): Given a complete undirected graph $G=(V,E)$ with $|V|=n$ and a weight d_{ij} on edge (i,j), $i<j$. Find a partition of V into two sets V_1 and $V\setminus V_1$ such that the sum of weights of edges with one endpoint in V_1 and the other in $V\setminus V_1$ is maximized. Formally, let $x_i = 1$ ($x_i = 0$) if $i \in V_1$ ($i \in V\setminus V_1$) and $\bar{x}_i = 1 - x_i$. Define $x_i \oplus x_j = \bar{x}_i x_j + x_i \bar{x}_j$, i.e., $x_i \oplus x_j = 0$ if and only if $x_i = x_j$. The problem is to find a vector $x \in X$ such that

$$\sum_{1\leqslant i<j\leqslant n} d_{ij} x_i \oplus x_j$$

is maximized.

Minimum weighted clique in a complete graph (MinClique): Given a complete undirected graph $G=(V,E)$ with $|V|=n$ and weight $d_{ij}=d_{ji}$ on edge (i,j), $1 \leqslant i,j \leqslant n$ (note that the graph contains self-loops, i.e., edges of the form (i,i)). Find a subset V_1 of nodes such that the sum of weights of edges with both endpoints in V_1 is minimized.

Formally, let $x_i = 1$ ($x_i = 0$) if $i \in V_1$ ($i \notin V_1$). The problem is to find a vector $x \in X$ such that

$$\sum_{1 \leqslant i,j \leqslant n} d_{ij} x_i x_j$$

is minimized.

The following lemmas give polynomial reductions between the problems just defined. If problem $P1$ polynomially reduces to problem $P2$, we write "$P1 \propto P2$". For an introduction to polynomial reducibility, see [6].

Lemma 1. *MaxCut* $\propto$ *MinClique.*

Proof. See the appendix.

Lemma 2. (a) *MAKS* $\propto$ *MaxCut.*
(b) *WCT* $\propto$ *MaxCut.*
(c) *CTV* $\propto$ *MaxCut.*
(d) *WET* $\propto$ *MinClique.*

Proof. See the appendix.

Note that, due to the transitivity of polynomial reducibility, all the problems that polynomially reduce to MaxCut also polynomially reduce to MinClique. It is interesting to note that WET does not reduce to MaxCut; the objective function of MaxCut is symmetric, i.e., the total weight of cut $V_1, V \setminus V_1$ is the same as the total weight of cut $V \setminus V_1, V_1$. This is not the case for WET; for a given schedule swapping all jobs finished by due date with the jobs finished after it may change total weighted earliness/tardiness.

3. Dynamic programming algorithms for MinClique

The proofs of Lemmas 1 and 2 show that for MAKS, WCT, and WET the outcome of reduction to MinClique is a graph with its weights being of the form $d_{ij} = d_{ji} = a_i b_j$ for $1 \leqslant j < i \leqslant n$ and $d_{ii} = d_i$ for $1 \leqslant i \leqslant n$. This also holds for CTV [16]. This section exploits the form in developing dynamic programming algorithms for this special case of MinClique. Without loss of generality, we can consider MinClique in the following form: Given matrix $D = (d_{ij})$, $1 \leqslant j \leqslant i \leqslant n$. Find vector $x \in X$ such that

$$\sum_{1 \leqslant j \leqslant i \leqslant n} d_{ij} x_i x_j$$

is minimized.

This problem is a non-linear program in 0–1 variables. Hansen et al. [9] present an excellent review of general methods for solving non-linear 0–1 programs.

First, we will present the row-part of the (r-c) pair of dynamic programming algorithms. Define

$$h_k(x) = \sum_{\substack{1 \leqslant j \leqslant i \leqslant n \\ k \leqslant i}} d_{ij} x_i x_j \tag{1}$$

for $k = 1, \ldots, n$, and

$$h(k, B) = \min_{x \in J(k,B)} \{h_k(x)\},$$

where $J(k, B) = \{x \in X : \sum_{j=1}^{k-1} b_j x_j = B\}$. From (1), we have

$$\begin{aligned} h_k(x) &= h_{k+1}(x) + \sum_{j=1}^{k} d_{kj} x_k x_j \\ &= h_{k+1}(x) + \sum_{j=1}^{k-1} d_{kj} x_k x_j + d_k x_k \\ &= h_{k+1}(x) + a_k x_k \sum_{j=1}^{k-1} b_j x_j + d_k x_k. \end{aligned} \tag{2}$$

From (1) and (2), we get the following recurrence relation:

$$\begin{aligned} h(k, B) &= \min_{x \in J(k,B)} \{h_k(x)\} \\ &= \min_{x \in J(k,B)} \left\{ h_{k+1}(x) + a_k x_k \sum_{j=1}^{k-1} b_j x_j + d_k x_k \right\} \\ &= \min \left\{ \min_{x \in J(k+1,B)} \{h_{k+1}(x)\}, \min_{x \in J(k+1,B+b_k)} \{h_{k+1}(x) + a_k B + d_k\} \right\} \qquad (3) \\ &= \min\{h(k+1, B), h(k+1, B+b_k) + a_k B + d_k\}. \qquad (4) \end{aligned}$$

We obtain Eq. (3) by considering the cases $x_k = 0$ (the first alternative) and $x_k = 1$ (the second alternative). The recurrence (4) should be solved for $h(1, 0)$, with the initial condition $h(n+1, B) = 0$ for all B. Since $0 \leqslant B \leqslant \sum_{i=1}^{n} b_i$ and $1 \leqslant k \leqslant n+1$, the solution can be found in time $O(n \sum_{i=1}^{n} b_i)$.

Now, we will present the column-part. Define

$$g_k(x) = \sum_{\substack{1 \leqslant j \leqslant i \leqslant n \\ j \leqslant k}} d_{ij} x_i x_j \tag{5}$$

for $k = 1, \ldots, n$, and

$$g(k, A) = \min_{x \in I(k,A)} \{g_k(x)\},$$

where $I(k,A) = \{x \in X : \sum_{i=k+1}^{n} a_i x_i = A\}$. From (5), we have

$$\begin{aligned} g_k(x) &= g_{k-1}(x) + \sum_{i=k}^{n} d_{ik} x_i x_k \\ &= g_{k-1}(x) + \sum_{i=k+1}^{n} d_{ik} x_i x_k + d_k x_k \\ &= g_{k-1}(x) + b_k x_k \sum_{i=k+1}^{n} a_i x_i + d_k x_k. \end{aligned} \tag{6}$$

From (5) and (6), we get the following recurrence relation:

$$\begin{aligned} g(k,A) &= \min_{x \in I(k,A)} \{g_k(x)\} \\ &= \min_{x \in I(k,A)} \left\{ g_{k-1}(x) + b_k x_k \sum_{i=k+1}^{n} a_i x_i + d_k x_k \right\} \\ &= \min \left\{ \min_{x \in I(k-1,A)} \{g_{k-1}(x)\}, \min_{x \in I(k-1,A+a_k)} \{g_{k-1}(x) + b_k A + d_k\} \right\} \\ &= \min\{g(k-1,A), g(k-1,A+a_k) + b_k A + d_k\} \end{aligned} \tag{7}$$

The recurrence (7) should be solved for $g(n,0)$, with the initial condition $g(0,A) = 0$ for all A. The solution can be found in time $O(n \sum_{i=1}^{n} a_i)$.

This (r-c) pair of dynamic programs very naturally translates into $O(n \sum_{i=1}^{n} p_i)$ and $O(n \sum_{i=1}^{n} w_i)$ pseudopolynomial time dynamic programs for WET and WCT. This implies that swapping processing times of all jobs with their weights results in no efficiency improvement. This result has been proven by De et al. [4] for WET, however, we are not aware of any such result for WCT. An (r-c) pair for CTV is given by Kubiak [16].

4. Spectral algorithms

This section derives spectral algorithms for MinClique (Sections 4.1 and 4.2). Spectral algorithms for MaxCut are briefly discussed in Section 4.3. The Hoffman–Wielandt inequality [10, Theorem 6.3.5 pp. 368–369, Exercise 7, p. 376] is central to the approach presented here. The inequality states that the distance of two $n \times n$ matrices A and B measured by the Frobenius norm (also called Euclidean norm) of their difference fulfills the following condition:

$$\sum_{i=1}^{n} (\mu_i - \lambda_{\sigma(i)})^2 \leqslant \|A - B\|^2 \leqslant \sum_{i=1}^{n} (\mu_i - \lambda_{\tau(i)})^2$$

for some permutations σ, τ of $\{1,2,\ldots,n\}$ where μ_i, λ_i are the eigenvalues of A and B, respectively. We shall apply this inequality to the distance between the adjacency

matrix D of graph G and the matrix

- $P(x) = (p_{ij}(x) = x_i x_j)$ (Section 4.1)
- $Q(x) = (q_{ij}(x) = 1 - x_i x_j)$ (Section 4.2),

respectively, where $x \in X$ is the indicator vector of a clique.

4.1. P-matrix algorithm

We apply the Hoffman–Wielandt inequality to matrices D and $P(x)$. Eigenvalues and eigenvectors of matrix $P(x)$ are given by the following lemma.

Lemma 3. *Let $0 < m \leqslant n$, and $x \in X$ with $m = |M|$ where $M = \{i : x_i = 1\}$. Then, the matrix $P(x)$ defined by $P(x) = (p_{ij}(x) = x_i x_j)$, $1 \leqslant i, j \leqslant n$ has only one non-zero eigenvalue $\lambda = m$. The normalized eigenvector $(v_1, v_2, \ldots, v_n)^{\mathrm{T}}$ associated with λ is given by $v_i = (s/\sqrt{m})x_i$ $(i = 1, \ldots, n)$ with $s \in \{-1, 1\}$.*

Proof. See the appendix.

Let $x \in X$, and $M = \{i : x_i = 1\}$, $m = |M|$, $m > 0$. We have the following equation:

$$\|D - P(x)\|^2 = \|D\|^2 - 2\sum_{i,j} d_{ij} p_{ij}(x) + \|P(x)\|^2$$
$$= \|D\|^2 + m^2 - 2\sum_{i,j} d_{ij} x_i x_j.$$

This implies that the problem of finding a minimum weighted clique of size m in a complete graph is equivalent to the problem of finding a vector $x \in X$ with $|\{i : x_i = 1\}| = m$ that maximizes $\|D - P(x)\|^2$. On the other hand, applying the RHS of the Hoffman–Wielandt inequality to matrices D and $P(x)$ and respecting Lemma 3, we obtain

$$\|D - P(x)\|^2 \leqslant (\mu_r - m)^2 + \sum_{\substack{i=1 \\ i \neq r}}^{n} \mu_i^2$$

for some $1 \leqslant r \leqslant n$, where $\mu_1, \mu_2, \ldots, \mu_n$ are the eigenvalues of matrix D. Notice that, since there is only one non-zero eigenvalue of matrix $P(x)$, there are only n possible choices for r. Let $U = (u_1, \ldots u_n)$, where $u_j = (u_{ij}), i = 1, \ldots, n$ be the orthonormal eigenvectors of matrix D associated with the eigenvalues $\mu_1, \ldots, \mu_n$, respectively, and v be the eigenvector of matrix $P(x)$. Let $M = \mathrm{diag}(\mu_1, \ldots, \mu_n)$. We have the following decompositions of D and $P(x)$

$$D = UMU^{\mathrm{T}}, \qquad P(x) = vmv^{\mathrm{T}}$$

since $U^{\mathrm{T}}U = I$ and $v^{\mathrm{T}}v = 1$ where I denotes the $n \times n$ identity matrix. This gives

$$\|D - P(x)\|^2 = \|UMU^{\mathrm{T}} - vmv^{\mathrm{T}}\|^2 = \|M - (U^{\mathrm{T}}v)m(U^{\mathrm{T}}v)^{\mathrm{T}}\|^2,$$

where the last inequality holds because matrix U is unitary. If we could find vector v such that $U^{\mathrm{T}}v = J$ where $J = (j_1, \ldots, j_n)$ is an n-element vector with $j_r = 1$ and all other entries equal to zero we would have equality

$$\|D - P(x)\|^2 = (\mu_r - m)^2 + \sum_{\substack{i=1 \\ i \neq r}}^{n} \mu_i^2,$$

and thus $\|D-P(x)\|^2$ would attain maximum. Notice that the LHS of the above equation depends on vector x, whereas the RHS does not as long as x chooses m nodes from set V. However, it may be impossible to find such vector v. Thus, we look for vector v such that $\|U^{\mathrm{T}}v - J\|$ is minimized. We have

$$\begin{aligned}\|U^{\mathrm{T}}v - J\|^2 = \|v - UJ\|^2 &= \sum_{i=1}^{n} (v_i - u_{ir})^2 \\ &= \sum_{i=1}^{n} v_i^2 - 2\sum_{i=1}^{n} v_i u_{ir} + \sum_{i=1}^{n} u_{ir}^2 \\ &= 2 - 2\frac{s}{\sqrt{m}} \sum_{i=1}^{n} u_{ir} x_i.\end{aligned}$$

Our first heuristic enumerates all possible triples (s,r,m) and for each of them finds vector $x \in X$ that maximizes $\sum_{i=1}^{n} u_{ir}x_i$ (and, thus, minimizes $\|U^{\mathrm{T}}v - J\|^2$).

Algorithm 1

Step 1 Find the orthonormal eigenvectors $u_j, j = 1, \ldots, n$ of D.

Step 2 For every (s,r,m) solve the following problem:

$$\begin{aligned}P(s,r,m): \max\ & s \sum_{i=1}^{n} u_{ir} x_i \\ \text{s.t.}\ & \sum_{i=1}^{n} x_i = m \\ & x_i \in \{0,1\} \quad (i = 1, \ldots, n)\end{aligned}$$

Observe that $\max\{P(1,r,m)\} = \min\{P(-1,r,m)\}$ so only $s = 1$ has to be considered.

Thus, program P in Step 2 is solved n^2 times. The solution involves finding either m largest or m smallest elements u_{ir} of the r's eigenvector u_r of matrix D. This can be done in linear time [3], so Step 2 takes $O(n^3)$ time.

4.2. *Q-matrix algorithm*

Now, we apply the Hoffman–Wielandt inequality to matrices D and $Q(x)$. We have the following lemma.

Lemma 4. *Let $0 < m < n$, and $x \in X$ with $m = |M|$, where $M = \{i : x_i = 1\}$. Then, the matrix $Q(x)$ defined by $Q(x) = (q_{ij}(x) = 1 - x_i x_j)$, $1 \leqslant i,j \leqslant n$ has only two different non-zero eigenvalues $\lambda_{1,2} = \frac{1}{2}((n-m) \pm \sqrt{(n+3m)(n-m)})$. The orthonormalized eigenvectors $v^1 = (v_i^1), v^2 = (v_i^2)$ associated with λ_1, λ_2 are given by*

$$v_i^1 = s_1 \frac{\lambda_1 + \lambda_2 x_i}{(n-m)\sqrt{2m+\lambda_1}},$$

$$v_i^2 = s_2 \frac{\lambda_2 + \lambda_1 x_i}{(n-m)\sqrt{2m+\lambda_2}}$$

$(i = 1, \ldots, n)$ with $s_1, s_2 \in \{-1, 1\}$.

Proof. See the appendix.

The following equation shows that the problem of finding a minimum weighted clique of size m in a complete graph is equivalent to the problem of finding a vector $x \in X$ with $|\{i : x_i = 1\}| = m$ minimizing $\|D - Q(x)\|^2$:

$$\begin{aligned}\|D - Q(x)\|^2 &= \|D\|^2 - 2\sum_{i,j} d_{ij} q_{ij} + \|Q(x)\|^2 \\ &= \|D\|^2 + n^2 - m^2 - 2\sum_{i,j} d_{ij} + 2\sum_{i,j} d_{ij} x_i x_j.\end{aligned}$$

On the other hand, the LHS of the Hoffman–Wielandt inequality yields

$$\|D - Q(x)\|^2 \geqslant (\mu_1 - \lambda_r)^2 + (\mu_2 - \lambda_t)^2 + \sum_{i \neq r,t} \mu_i^2$$

for some $1 \leqslant r, t \leqslant n$ and $r \neq t$. We are now looking for a clique x such that $\|D - Q(x)\|^2$ is minimized. The reasoning leading to Algorithm 1 in Section 4.1 can be repeated here for matrix $Q(x)$. Notice that now we have matrix $V = (v^1, v^2)$ of eigenvectors of $Q(x)$, thus $V \Lambda V^{\mathrm{T}} = Q(x)$ with $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2)$. Consequently, we are looking for V such that $U^{\mathrm{T}} V = J$, or, to minimize

$$\begin{aligned}\|U^{\mathrm{T}} V - J\|^2 = \|V - UJ\|^2 &= \sum_{i=1}^{n} (v_i^1 - u_{ir})^2 + \sum_{i=1}^{n} (v_i^2 - u_{it})^2 \\ &= \sum_{i=1}^{n} (v_i^1)^2 + \sum_{i=1}^{n} u_{ir}^2 + \sum_{i=1}^{n} (v_i^2)^2 + \sum_{i=1}^{n} u_{it}^2 - 2\sum_{i=1}^{n} (v_i^1 u_{ir} + v_i^2 u_{it}) \\ &= 4 - 2\sum_{i=1}^{n} \left(s_1 u_{ir} \frac{\lambda_1 + \lambda_2 x_i}{(n-m)\sqrt{2m+\lambda_1}} + s_2 u_{it} \frac{\lambda_2 + \lambda_1 x_i}{(n-m)\sqrt{2m+\lambda_2}} \right) \\ &= 4 - \frac{2}{n-m} \sum_{i=1}^{n} \left(\frac{s_1 u_{ir} \lambda_1}{\sqrt{2m+\lambda_1}} + \frac{s_2 u_{it} \lambda_2}{\sqrt{2m+\lambda_2}} \right) \\ &\quad - \frac{2}{n-m} \sum_{i=1}^{n} \left(\frac{s_1 u_{ir} \lambda_2}{\sqrt{2m+\lambda_1}} + \frac{s_2 u_{it} \lambda_1}{\sqrt{2m+\lambda_2}} \right) x_i.\end{aligned}$$

Thus, our goal is to find x maximizing

$$\sum_{i=1}^{n}\left(\frac{s_1 u_{ir}\lambda_2}{\sqrt{2m+\lambda_1}}+\frac{s_2 u_{it}\lambda_1}{\sqrt{2m+\lambda_2}}\right)x_i.$$

As a result we obtain Algorithm 2, below.

Algorithm 2

Step 1 As in Algorithm 1.

Step 2 For every (s_1, s_2, r, t, m) solve the following program:

$$Q(s_1,s_2,r,t,m): \max \sum_{i=1}^{n}\left(\frac{s_1 u_{ir}\lambda_2}{\sqrt{2m+\lambda_1}}+\frac{s_2 u_{it}\lambda_1}{\sqrt{2m+\lambda_2}}\right)x_i \tag{8}$$

$$\text{s.t.} \sum_{i=1}^{n} x_i = m$$

$$x_i \in \{0,1\} \quad (i=1,\ldots,n)$$

Note $\max Q(1,1,r,t,m) = \min Q(-1,-1,r,t,m)$, and $\max Q(1,-1,r,t,m) = \min Q(-1,1,r,t,m)$, i.e., only two pairs of (s_1,s_2) (namely, $(1,1)$ and $(1,-1)$) have to be considered. Thus, program Q is solved $O(n^3)$ times. The solution involves finding either m largest or m smallest coefficients of x_i's in (8). This can be done in linear time, so Step 2 takes $O(n^4)$ time.

4.3. Spectral algorithms for MaxCut

As in Section 4.1, it can be shown that the problem of finding a maximum weighted cut of size m in a complete graph is equivalent to the problem of finding vector $x \in X$ with $|\{i : x_i = 1\}| = m$ maximizing $\|D - P'(x)\|^2$ with $P'(x) = (p'_{ij}(x)) = (1 - x_i \oplus x_j)$. Thus, we obtain the following algorithm [11].

Algorithm 3

Step 1 As in Algorithm 1.

Step 2 For every (s_1, s_2, r, t, m) solve the following program:

$$P'(s_1,s_2,r,t,m): \max \sum_{i=1}^{n}\left(\frac{s_1 u_{ir}}{\sqrt{m}}-\frac{s_2 u_{it}}{\sqrt{n-m}}\right)x_i \tag{9}$$

$$\text{s.t.} \sum_{i=1}^{n} x_i = m$$

$$x_i \in \{0,1\} \quad (i=1,\ldots,n)$$

Again, the solution involves finding either m largest or m smallest coefficients of x_i's in (9), so Step 2 takes $O(n^4)$ time. By neglecting the square roots in Step 2, we can

reduce the time spent in Step 2 by the factor n. As result, we obtain Algorithm 4, below.

Algorithm 4

Step 1 As in Algorithm 1.

Step 2 For every (s_1, s_2, r, t) solve the following program:

$$P''(s_1,s_2,r,t): \max \sum_{i=1}^{n} (s_1 u_{ir} - s_2 u_{it}) x_i$$

$$\text{s.t.} \sum_{i=1}^{n} x_i = m$$

$$x_i \in \{0,1\} \quad (i = 1,\ldots,n)$$

4.4. *Applying spectral algorithms to MinClique problems*

In order to apply a spectral algorithm to an instance of a MinClique scheduling problem, matrix D is computed for the instance first. The formulas for the entries of D are given in the proofs of Lemmas 1 and 2 (see the appendix). The algorithm then produces a 0–1 vector that can be translated into a solution to the problem. Again, the easy translation is given in the proofs of Lemmas 1 and 2 (see the appendix).

5. Improvement procedures

We have found that the spectral algorithms defined in Section 4 are likely to produce solutions which are not local optima, though they are very receptive to simple improvements. We used two different types of improvement procedures:

- a simple job exchange procedure and
- a tabu search procedure.

The simple job exchange procedure passes through the list of jobs (ordered according to SPT rule in the case of MAKS and CTV and according to Smith's rule in the case of WCT and WET) and considers all pairs $(i, i+1)$, $1 \leqslant i \leqslant n-1$. If $x_i \neq x_{i+1}$, then it calculates the objective value of the schedule obtained by taking $\bar{x}_i$ and $\bar{x}_{i+1}$. If this value is lower than the one for the original schedule, then the jobs are exchanged and the next pair $(i+1, i+2)$ is considered; otherwise, no exchange is made before passing to the next pair. The procedure passes through the list of all jobs not more than a fixed number of times; it also stops if a pass does not improve a schedule.

Our tabu search algorithm follows the general framework presented in [7], where a tabu search algorithm starts from some initial solution and searches iteratively through the set X until some stop condition is fulfilled. This framework admits a large degree of flexibility in defining neighborhood, tabu list, and stop condition. These parameters for our tabu search algorithm are as follows.

Neighborhood: Let $x^k=(x_1^k,\ldots,x_n^k)\in X$ be the initial solution of iteration k. The neighborhood of x^k defines a set of solutions which can become initial solutions in iteration $k+1$. We define it as follows:

Let $\text{last}(j)=\max\{i<k : x_j^i \neq x_j^{i+1}\}$ (i.e., $\text{last}(j)$ is the index of the most recent iteration when a change to x_j occurred), $j\in\{1,\ldots,n\}$. Let $I=\{j : k-\text{last}(j) > \lfloor n2^{-s}\rfloor\}$ for some given integer s between 1 and n. We define

$$x'\in N_s(x^k) \Leftrightarrow \exists J\subseteq I, |J|\leqslant s \text{ such that } x'_j=1-x_j \text{ for all } j\in J \text{ and } x'_j=x_j \text{ for all } j\in\{1,\ldots,n\}\setminus J.$$

Next solution: To choose the initial solution for iteration $k+1$, we consider the neighborhood $N_s(x^k)$ as well as the extended neighborhood $N_s^*(x^k)$ defined as follows:

$$x'\in N_s^*(x^k) \Leftrightarrow \exists J\subseteq\{1,\ldots,n\}, |J|\leqslant s \text{ such that } x'_j=1-x_j \text{ for all } j\in J \text{ and } x'_j=x_j \text{ for all } j\in\{1,\ldots,n\}\setminus J.$$

Let x' and x'' be the best solutions in $N_s^*(x^k)$ and $N_s(x^k)$, respectively. If x' improves the best solution found so far, we choose it as an initial solution for the next iteration. Otherwise, we choose x''.

Stop condition: The algorithm stops if either

- no initial solution for the next iteration can be found, i.e., $N_s(x^k)=\emptyset$ and no solution in $N_s^*(x^k)$ improves the best solution found so far, or
- the best solution has not been improved for n iterations, where n is the number of jobs.

6. Computational results

We tested two approaches to obtaining suboptimal solutions to the MinClique scheduling problems. One gives any solution obtained by the spectral algorithms (the output of each spectral algorithm is a *set* of solutions, one for each (s,r,m) (or (s_1,s_2,r,t,m))) a chance of improvement by applying to it the job exchange procedure (see algorithms SkE below). The other gives this chance to the best of them only, however, the improvement algorithm, being a tabu search algorithm (see algorithms SkT below), is more sophisticated than the job exchange procedure. All algorithms tested were implemented in FORTRAN 77 on DEC 5500 computer. We used standard IMSL procedures to generate random numbers and calculate eigenvectors. We will present results obtained for problems MAKS, WCT, CTV, and WET in Sections 6.1–6.4, respectively. Throughout these subsections, we use the following notations:

n the number of jobs

#pro the number of instances tested for a given problem size

SkE Algorithm k ($k \in 1,\ldots,4$) with the job exchange procedure applied to *all* solutions

SkT Tabu search with initial solution obtained by choosing the *best* solution generated by Algorithm k ($k \in 1,\ldots,4$)

OPT complete enumeration procedure

We tested all spectral algorithms for all problems, though we only report results of the most accurate. To reduce computational times of Algorithms S2E, S2T, S3E, and S3T, we considered values m in the interval $\lfloor \lceil n/2 \rceil - \log n \rfloor \leqslant m \leqslant \lceil n/2 \rceil$ only.

6.1. *MAKS*

Instances with $n = 10, 15, \ldots, 70$ jobs were randomly generated with processing times uniformly distributed over the interval $[1, 2^n]$. (Problems with processing times uniformly distributed over the interval $[1, 2^{n/2}]$ turned out to be likely solved to optimality by the spectral algorithms for MAKS.) The results are presented in Tables 1 and 2. We use the following notations:

LPTT Tabu search, with initial solution obtained by the LPT algorithm

KKT Tabu search, with initial solution obtained by the differencing algorithm of Karmakar and Karp [14]

We used neighborhood N_3, i.e., $s=3$, for all tabu search algorithms. The job exchange procedure was applied at most three times. Table 1 shows the maximum relative error over all test runs for all the algorithms tested, i.e., the maximum of all values

$$\frac{\text{MAKS(alg)} - \text{MAKS(best)}}{\text{MAKS(best)}},$$

where MAKS(best) equals optimal value, for $n \leqslant 25$, and equals the best solution obtained by the six heuristics, for $n > 25$. Optimal solutions for $n \leqslant 25$ were obtained by a complete enumeration procedure. Table 2 gives average computational times.

Table 1
MAKS. Maximum relative error $* 10^{10}$

n	#pro	LPTT	KKT	S2E	S2T	S3E	S3T
10	20	1 283 6970	1 283 6970	0	0	0	0
15	20	3 933 874	4 001 088	734 545	3 275 168	734 545	3 682 670
20	20	431 066	280 461	192 859	409 188	134 332	409 188
25	20	249 199	112 729	65 944	97 724	65 944	71 048
30	20	92 716	63 302	24 377	39 903	18 176	66 041
35	20	12 859	42 575	4971	53 966	13 955	12 761
40	20	21 174	7886	2116	23 499	3678	16 833
45	20	5050	4457	2101	4772	3599	4772
50	20	2128	3211	2035	2390	2332	2390
55	10	1250	1411	682	4696	588	752
60	10	502	554	985	1244	985	1840
65	10	422	729	215	1051	401	625
70	10	397	882	192	249	548	319

Table 2
Computational times for MAKS (in s)

n	LPTT	KKT	S2E	S2T	S3E	S3T	OPT
10	0.01	0.01	0.16	0.10	0.09	0.05	0.00
15	0.05	0.05	0.54	0.35	0.29	0.20	0.07
20	0.17	0.18	1.63	1.08	0.82	0.64	2.11
25	0.46	0.39	3.13	2.17	1.68	1.34	66.64
30	0.86	0.88	5.36	3.76	2.79	2.40	—
35	1.71	1.61	10.30	7.30	5.29	4.64	—
40	2.53	2.26	15.70	10.89	7.77	6.88	—
45	4.53	4.85	23.38	17.71	11.86	11.29	—
50	6.19	5.94	30.00	22.74	15.20	15.30	—
55	9.25	8.46	39.63	31.07	19.99	22.94	—
60	15.76	12.15	51.34	41.46	26.28	27.58	—
65	18.23	18.87	76.88	63.05	39.16	38.92	—
70	23.33	18.85	92.87	72.26	47.54	51.72	—

Table 3
MAKS. Maximum percentage of improvement

n	#pro	S2E	S2T	S3E	S3T
10	20	0.592541	0.592541	0.383409	0.383409
15	20	0.212350	0.214039	0.224388	0.224388
20	20	0.060653	0.059897	0.057420	0.050638
25	20	0.037051	0.036983	0.016958	0.016820
30	20	0.016389	0.016510	0.011385	0.011182
35	20	0.029420	0.029405	0.010632	0.010664
40	20	0.006710	0.006720	0.006710	0.006720
45	20	0.006536	0.006496	0.004655	0.004657
50	20	0.006095	0.006110	0.003746	0.003747
55	10	0.003393	0.003390	0.011489	0.011495
60	10	0.003298	0.003282	0.000638	0.000633
65	10	0.001880	0.001880	0.001220	0.001210
70	10	0.001560	0.001560	0.001700	0.001700

Algorithms S2E and S3E are more accurate than the tabu search algorithms at the cost of longer computational times. For all problem sizes, except $n=60$, the maximum relative error of the more accurate of a pair S2E and S3E is less than the error of the most accurate tabu search algorithm. KKT is more accurate for smaller instances, while LPTT is for larger. The calculation of initial solutions takes longer for S2T and S3T than for LPTT and KKT.

Table 3 presents the maximum percentage of improvement obtained by the exchange procedure and the tabu search. More precisely, columns SkE, $k=2$ and 3, give the maximum of

$$\frac{\mathrm{MAKS(Sk)} - \mathrm{MAKS(SkE)}}{\mathrm{MAKS(SkE)}} * 100$$

over all instances for given n, where MAKS(Sk) is value of the best solution generated by Algorithm k. Columns SkT, $k = 2$ and 3, give the maximum of

$$\frac{\text{MAKS(Sk)} - \text{MAKS(SkT)}}{\text{MAKS(SkT)}} * 100$$

over all instances for given n.

6.2. WCT

Instances with $n = 10, 15, \ldots, 60$ jobs were randomly generated with processing times and weights uniformly distributed over the interval $[1, 2^{n/2}]$. (Note that the weighted completion time of a schedule is usually much larger than its makespan, so larger data very likely produce data overflow.) The results are presented in Tables 4 and 5, where SMIT denotes tabu search with initial solution obtained by the algorithm of Smith [23]. We used neighborhood N_3 for all tabu search algorithms. Table 4 shows the maximum

Table 4
WCT. Maximum relative error $* 10^{10}$

n	#pro	SMIT	S1T	S4T
10	20	0	0	0
15	20	489 626	345 761	345761
20	20	1 972 638	2 961 063	2 825 877
25	20	2 004 996	1 692 142	1 692 142
30	20	1 140 218	6 454 808	2 971 961
35	20	1 533 567	3 128 518	750 714
40	20	1 734 412	2 026 924	1 713 113
45	20	1 052 836	1 582 437	1 052 836
50	20	838 727	544 944	481 528
55	10	477 476	673 233	238 049
60	10	235 091	342 848	317 208

Table 5
Computational times for WCT (in s)

n	SMIT	S1T	S4T	OPT
10	0.04	0.05	0.05	0.01
15	0.25	0.28	0.32	0.41
20	0.99	1.21	1.28	16.49
25	3.21	3.61	3.70	—
30	7.11	9.12	8.89	—
35	20.31	20.75	19.52	—
40	33.61	37.14	41.97	—
45	69.53	64.31	67.38	—
50	109.41	122.62	120.64	—
55	179.08	166.91	158.26	—
60	248.81	286.72	294.79	—

relative error over all test runs for all algorithms tested in comparison with the best solution found, i.e., the maximum of all values

$$\frac{\mathrm{WCT(alg)} - \mathrm{WCT(best)}}{\mathrm{WCT(best)}},$$

where WCT(best) equals optimal value, for $n \leqslant 20$, and equals the best solution obtained by the three heuristics for, $n > 20$. Optimal solutions for $n \leqslant 20$ were obtained by a complete enumeration procedure. Table 5 gives average computational times.

Algorithm S4T is more accurate than SMIT and S1T for almost all problem sizes. Since the differences between computational times of all three algorithms are negligable, we may conclude that S4T proved to be the best algorithm for WCT.

Table 6 presents the maximum percentage of improvement obtained by the tabu search. More precisely, columns SkT, $k = 1$ and 4, give the maximum of

$$\frac{\mathrm{WCT(Sk)} - \mathrm{WCT(SkT)}}{\mathrm{WCT(SkT)}} * 100$$

over instances for given n, where WCT(Sk) is value of the best solution generated by Algorithm k.

6.3. CTV

Instances with $n = 10, 15, \ldots, 70$ jobs were randomly generated with processing times uniformly distributed over the interval $[1, 2^{n/4}]$. The results are presented in Tables 7 and 8, where KANT denotes tabu search with initial solution obtained by the algorithm given by Kanet (1981). We used neighborhood N_2 instead of N_3 for all tabu search algorithms because of long computational times. The job exchange procedure was applied only twice for the same reason. Table 5 shows the maximum relative error over all test runs for all algorithms tested in comparison with the best solution found, i.e.,

Table 6
WCT. Maximum percentage of improvement

n	#pro	S4T	S1T
10	20	0.572387	0.948570
15	20	0.641231	0.923253
20	20	0.716697	1.042383
25	20	0.672731	1.032177
30	20	0.654934	1.109232
35	20	0.576160	0.792774
40	20	0.510820	0.715477
45	20	0.518730	0.785954
50	10	0.514640	0.561260
55	10	0.772597	0.535167
60	10	0.403214	0.731112

Table 7
CTV. Maximum relative error $* 10^{10}$

n	#pro	KANT	S3E	S3T	S4T
10	20	0	0	0	0
15	20	0	0	0	0
20	20	0	0	79156	6824
25	20	56344	28012	587847	86725
30	20	85444	4341	58780	214600
35	20	23107	13202	73084	19526
40	10	867	3876	33262	33262
45	10	976	34008	32165	11929
50	10	11695	12301	14545	7895
55	10	3623	—	11024	13170
60	10	418	—	8798	9193
65	10	256	—	6308	4595
70	10	3035	—	3776	10042

Table 8
Computational times for CTV (in s)

n	KANT	S3E	S3T	S4T	OPT
10	0.02	0.87	0.06	0.03	0.04
15	0.11	5.34	0.23	0.15	1.04
20	0.38	21.33	0.70	0.45	37.83
25	0.92	52.20	1.41	1.16	—
30	1.83	115.81	2.64	2.66	—
35	3.51	254.52	4.97	4.84	—
40	5.37	410.48	7.23	7.29	—
45	8.73	633.33	9.94	11.86	—
50	14.22	1042.66	15.64	21.86	—
55	22.22	—	22.10	29.37	—
60	30.03	—	26.13	36.67	—
65	42.68	—	38.27	61.23	—
70	47.67	—	46.08	65.65	—

the maximum of all values

$$\frac{\text{CTV(alg)} - \text{CTV(best)}}{\text{CTV(best)}},$$

where CTV(best) equals the optimal value, for $n \leqslant 20$, and equals the best solution obtained by the heuristics, for $n > 20$. Optimal solutions for $n \leqslant 20$ were obtained by a complete enumeration procedure. Table 8 gives average computational times.

Algorithm S3E gives the best results for small instances ($n \leqslant 35$); for larger instances, however, it is always less accurate than any of the tabu search algorithms. For large instances ($n \geqslant 55$), KANT always gives the most accurate solution. Differences between computational times of all tabu search algorithms are negligable; S4T is just a little more time consuming than KANT and S3T. The CPU-time of S3E grows quite fast with the number of jobs, thus, it has not been tested for instances with $n \geqslant 55$.

Table 9
CTV. Maximum percentage of improvement

n	#pro	S4T	S3E	S3T
10	20	0.060677	0	0
15	20	0.322411	0.000183	0.058725
20	20	0.616270	0.000233	0.148281
25	20	0.619933	0.000266	0.253635
30	20	0.717965	0.000101	0.387509
35	20	0.651513	0.000246	0.336413
40	10	0.694755	0.000249	0.405821
45	10	0.602000	0.369647	0.369627
50	10	0.742525	0.528303	0.528303
55	10	0.636135	-	0.468103
60	10	0.594127	-	0.451729
65	10	0.655024	-	0.425462
70	10	0.509770	-	0.374574

Table 9 presents the maximum percentage of improvement obtained by the exchange procedure and the tabu search. More precisely, column S3E gives the maximum of

$$\frac{\text{CTV(S3)} - \text{CTV(S3E)}}{\text{CTV(S3E)}} * 100$$

over all instances for given n, where CTV(Sk) is value of the best solution generated by Algorithm k. Columns SkT, $k = 3$ and 4, give the maximum of

$$\frac{\text{CTV(Sk)} - \text{CTV(SkT)}}{\text{CTV(SkT)}} * 100$$

over all instances for given n.

6.4. *WET*

Instances with $n = 10, 15, \ldots, 60$ jobs were randomly generated with processing times and weights uniformly distributed over the interval $[1, 2^{n/2}]$. We tested two different algorithms: S1T and tabu search with initial solution obtained by the modified Smith's algorithm (denoted by SMMT). The modified Smith's algorithm works as follows: Let the jobs be ordered such that $w_1/p_1 \geqslant w_2/p_2 \geqslant \ldots \geqslant w_n/p_n$ (see proof of Lemma 2 (d)). Assume that jobs $1, \ldots, i-1$ have already been scheduled. Let E and T denote the set of jobs in $\{1, \ldots, i-1\}$ scheduled early (i.e., by due date d) and late (i.e., after d), respectively. Initially, we have $E = \{k\}$ and $T = \{1, \ldots, k-1\}$ for some job $k < i$. If $\sum_{j \in E} p_j < \sum_{j \in T} p_j + p_i$, we add i to E; otherwise, we add it to T. The algorithm tries all the possible choices $k \in \{1, \ldots, n\}$ for the job finishing at d.

We used neighborhood N_3 for both tabu search algorithms. For $n \leqslant 20$, we calculated optimal solutions using a complete enumeration procedure. Both algorithms found the optimal solution for all instances with $n \leqslant 20$. For $n = 50$, S1T improves SMMT in one case out of 20. For $n = 60$, each algorithm improves the other in one case out

Table 10
Computational times for WET (in s)

n	SMMT	S1T	OPT
10	0.04	0.05	0.02
15	0.25	0.28	0.83
20	1.05	1.09	34.82
25	2.97	3.06	—
30	7.13	7.31	—
35	15.10	15.95	—
40	30.99	31.46	—
45	51.93	51.21	—
50	84.76	83.00	—
55	129.70	132.41	—
60	223.19	239.99	—

Table 11
WET. Maximum percentage of improvement

n	#pro	S1T
10	20	1.458671
15	20	5.652539
20	20	3.963582
25	20	4.584041
30	20	4.642857
35	20	3.738318
40	20	3.096539
45	20	2.432432
50	20	2.419355
55	10	2.276945
60	10	3.043950

of 10. In all other cases, the algorithms were of the same accuracy. Table 10 gives average computational times.

Table 11 presents the maximum percentage of improvement obtained by the tabu search. More precisely, column S1T gives the maximum of

$$\frac{\mathrm{WET(S1)} - \mathrm{WET(S1T)}}{\mathrm{WET(S1T)}} * 100$$

over all instances for given n, where WET(S1) is value of the best solution generated by Algorithm 1.

7. Concluding remarks

We considered four well-known scheduling problems that reduce to the problem of finding minimum weighted clique in a complete graph with negative weights and

self-loops allowed. Each of the reductions results in a graph in which the weights associated with edges (i,j) are of the form $a_i * b_j$. We presented an (r-c) pair of dynamic programs for this type of MinClique problem which translates into a pseudopolynomial time pair of dynamic programs for each of the scheduling problems.

We also gave a family of spectral algorithms for the MinClique problem as well as for MaxCut. The algorithms exploit eigenvectors of the adjacency matrix of a graph. Computational tests have shown that these algorithms often produce solutions that are not local optima, though they are very receptive to simple improvement algorithms. This may indicate that the bounds given by the Hoffman–Wielandt inequality for these problems are not strong enough. Notice that due to the small number of different non-zero eigenvalues of P and Q matrices we were able to enumerate all the bounds. Our computational test were designed to compare two different approaches to improving suboptimal solutions. One approach applies a simple job exchange procedure to *all* generated solution, the other applies tabu search to the *best* solution only. The former approach turned out to be more accurate for MAKS and small size CTV problems, the latter was more accurate for WCT and WET. The tabu search did not use any diversification or intensification procedures that might improve its quality. This addition to the tabu search as well as the development of lower bounds for the four problems seem to be promising directions for further research.

An interesting open question is whether there exists a fully polynomial time approximation scheme for MinClique with weights being of the form $a_i * b_j$. Such a scheme is given for CTV by De et al. [5]. The algorithm given by Hall and Posner [8] for WET becomes a fully polynomial time approximation scheme only if the weights are bounded by a polynomial function of n. Then, however, an exact solution can be obtained in polynomial time (see [5] or Section 3). Thus, the question is also open for WET.

Appendix

Proof of Lemma 1. We show that, for a given instance of MaxCut given by integers d_{ij}, $1 \leqslant i < j \leqslant n$, we can define integers d'_{ij}, $1 \leqslant i,j \leqslant n$ such that

$$\sum_{1 \leqslant i < j \leqslant n} d_{ij} x_i \oplus x_j = - \sum_{1 \leqslant i,j \leqslant n} d'_{ij} x_i x_j.$$

Let $d_{ji} = d_{ij}$ for $1 \leqslant i < j \leqslant n$. We have

$$\begin{aligned}
\sum_{1 \leqslant i < j \leqslant n} d_{ij} x_i \oplus x_j &= \sum_{1 \leqslant i < j \leqslant n} d_{ij} (\bar{x}_i x_j + x_i \bar{x}_j) \\
&= \sum_{1 \leqslant i < j \leqslant n} d_{ij} x_j + \sum_{1 \leqslant i < j \leqslant n} d_{ij} x_i - \sum_{1 \leqslant i < j \leqslant n} 2 d_{ij} x_i x_j \\
&= \sum_{1 \leqslant i < j \leqslant n} d_{ij} x_j x_j + \sum_{1 \leqslant i < j \leqslant n} d_{ij} x_i x_i - \sum_{\substack{1 \leqslant i,j \leqslant n \\ i \neq j}} d_{ij} x_i x_j
\end{aligned}$$

$$= \sum_{i=2}^{n} \left(\sum_{j=1}^{i-1} d_{ji} \right) x_i x_i + \sum_{i=1}^{n-1} \left(\sum_{j=i+1}^{n} d_{ij} \right) x_i x_i - \sum_{\substack{1 \leqslant i,j \leqslant n \\ i \neq j}} d_{ij} x_i x_j$$

$$= \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} d_{ji} + \sum_{j=i+1}^{n} d_{ij} \right) x_i x_i - \sum_{\substack{1 \leqslant i < j \leqslant n \\ i \neq j}} d_{ij} x_i x_j$$

$$= - \sum_{1 \leqslant i,j \leqslant n} d'_{ij} x_i x_j$$

with

$$d'_{ij} = \begin{cases} d_{ij} & \text{if } i \neq j, \\ -\left(\sum_{k=1}^{i-1} d_{ki} + \sum_{k=i+1}^{n} d_{ik} \right) & \text{if } i = j. \end{cases}$$ □

Proof of Lemma 2. (a) In an optimal schedule, the jobs are scheduled on both machines without idle time. Define $x_i = 1$ ($x_i = 0$) if job i is processed on M_1 (M_2). Obviously, minimizing the maximum completion time is equivalent to maximizing

$$\left(\sum_{j=1}^{n} x_j p_j \right) \left(\sum_{i=1}^{n} \bar{x}_i p_i \right) = \sum_{j=1}^{n} \sum_{i=1}^{j-1} p_j p_i x_j \bar{x}_i + \sum_{j=1}^{n} p_j p_j x_j \bar{x}_j + \sum_{j=1}^{n} \sum_{i=j+1}^{n} p_j p_i x_j \bar{x}_i$$

$$= \sum_{1 \leqslant i < j \leqslant n} p_j p_i x_j \bar{x}_i + \sum_{1 \leqslant i < j \leqslant n} p_i p_j x_i \bar{x}_j$$

$$= \sum_{1 \leqslant i < j \leqslant n} p_i p_j x_i \oplus x_j$$

$$= \sum_{1 \leqslant i < j \leqslant n} d_{ij} x_i \oplus x_j$$

with $d_{ij} = p_i p_j$ ($1 \leqslant i < j \leqslant n$).

(b) Let the jobs be ordered such that $w_1/p_1 \geqslant w_2/p_2 \geqslant \cdots \geqslant w_n/p_n$. In an optimal schedule, the jobs on each machine are scheduled in increasing order of their indices without idle time [23]. For a given schedule, define $x_i = 1$ ($x_i = 0$) if job i is processed on M_1 (M_2). The objective function of this schedule is equal to

$$\sum_{j=1}^{n} w_j x_j \left(\sum_{i=1}^{j} p_i x_i \right) + \sum_{j=1}^{n} w_j \bar{x}_j \left(\sum_{i=1}^{j} p_i \bar{x}_i \right)$$

$$= \sum_{j=1}^{n} w_j x_j \left(\sum_{i=1}^{j} p_i (1 - \bar{x}_i) \right) + \sum_{j=1}^{n} w_j \bar{x}_j \left(\sum_{i=1}^{j} p_i (1 - x_i) \right)$$

$$= \sum_{1 \leqslant i \leqslant j \leqslant n} w_j(x_j + \bar{x}_j)p_i - \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i(\bar{x}_i x_j + x_i \bar{x}_j)$$

$$= \sum_{1 \leqslant i \leqslant j \leqslant n} w_j p_i - \sum_{1 \leqslant i < j \leqslant n} d_{ij} x_i \oplus x_j$$

with $d_{ij} = w_j p_i$ $(1 \leqslant i < j \leqslant n)$.

(c) See [16].

(d) Again, let the jobs be ordered such that $w_1/p_1 \geqslant w_2/p_2 \geqslant \cdots \geqslant w_n/p_n$. Hall and Posner [8] show that there always exists an optimal schedule with the following properties:

- One job finishes at due date d.
- The jobs finishing by d are scheduled in an order of decreasing indices without idle time inserted.
- The jobs finishing after d are scheduled in an order of increasing indices without idle time inserted.

For a given schedule, define $x_i = 1$ if job i finishes by d, i.e., $C_i \leqslant d$, and $x_i = 0$ if job i finishes after d, i.e., $C_i > d$. The objective function of this schedule can be written as

$$\sum_{j=1}^{n} w_j x_j \left(\sum_{i=1}^{j-1} p_i x_i \right) + \sum_{j=1}^{n} w_j \bar{x}_j \left(\sum_{i=1}^{j} p_i \bar{x}_i \right)$$

$$= \sum_{j=1}^{n} w_j x_j \left(\sum_{i=1}^{j-1} p_i x_i \right) + \sum_{j=1}^{n} w_j (1 - x_j) \left(\sum_{i=1}^{j} p_i (1 - x_i) \right)$$

$$= \sum_{1 \leqslant i < j \leqslant n} w_j\, p_i x_j x_i + \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i - \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i x_i$$

$$- \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i x_j + \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i x_j x_i$$

$$= \sum_{1 \leqslant i < j \leqslant n} 2w_j\, p_i x_j x_i + \sum_{i=1}^{n} \left(-p_i \sum_{j=i}^{n} w_j - w_i \sum_{j=1}^{i} p_j + w_i\, p_i \right) x_i x_i + \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i$$

$$= \sum_{1 \leqslant i < j \leqslant n} 2w_j\, p_i x_j x_i + \sum_{i=1}^{n} - \left(w_i \sum_{j=1}^{i-1} p_j + p_i \sum_{j=i}^{n} w_j \right) x_i x_i + \sum_{1 \leqslant i \leqslant j \leqslant n} w_j\, p_i$$

$$= \sum_{1 \leqslant i,j \leqslant n} d_{ij} x_i x_j + \sum_{1 \leqslant i \leqslant j \leqslant n} w_j p_i$$

with

$$d_{ij} = \begin{cases} w_j p_i & \text{if} \quad i < j, \\ w_i p_j & \text{if} \quad i > j, \\ -\left(w_i \sum_{k=1}^{i-1} p_k + p_i \sum_{k=i}^{n} w_k \right) & \text{if} \quad i = j. \end{cases} \quad \square$$

Proof of Lemma 3. Let $\lambda \neq 0$ be an eigenvalue of matrix $P(x)$, and let $v = (v_1, v_2, \ldots, v_n)^T$ be a normalized eigenvector associated with λ. By definition [10], we have

$$P(x)v = \lambda v \Leftrightarrow \sum_{j=1}^{n} x_i x_j v_j = \lambda v_i \quad (i = 1, \ldots, n).$$

We consider the following cases:

(i) $i \in M$, i.e., $x_i = 1$. We have

$$\lambda v_i = \sum_{j=1}^{n} x_i x_j v_j = \sum_{j=1}^{n} x_j v_j = \sum_{j \in M} v_j, \tag{10}$$

i.e., we have $v_i = k$ for all $i \in M$ and some k.

(ii) $i \notin M$, i.e., $x_i = 0$. We have

$$\lambda v_i = \sum_{j=1}^{n} x_i x_j v_j = 0,$$

i.e., we have $v_i = 0$ for all $i \notin M$.

Because of $v \neq 0$ we have $k \neq 0$, and, following (10),

$$\sum_{j \in M} v_j = mk \quad \text{and} \quad \sum_{j \in M} v_j = \lambda k,$$

i.e., $\lambda = m$. Normalizing v gives $v_i = (s/\sqrt{m})x_i$ for $i = 1, \ldots, n$ and $s \in \{-1, 1\}$. □

Proof of Lemma 4. Let $\lambda \neq 0$ be an eigenvalue of matrix $Q(x)$, and let $v = (v_1, v_2, \ldots, v_n)^T$ be a normalized eigenvector associated with λ. We have

$$Q(x)v = \lambda v \Leftrightarrow \sum_{j=1}^{n} (1 - x_i x_j) v_j = \lambda v_i \quad (i = 1, \ldots, n).$$

We consider the following cases:

(i) $i \in M$, i.e., $x_i = 1$. We have

$$\lambda v_i = \sum_{j=1}^{n} (1 - x_i x_j) v_j = \sum_{j=1}^{n} (1 - x_j) v_j = \sum_{j \notin M} v_j,$$

i.e., we have $v_i = k$ for all $i \in M$ and some k.

(ii) $i \notin M, x_i = 0$. We have

$$\lambda v_i = \sum_{j=1}^{n} (1 - x_i x_j) v_j = \sum_{j=1}^{n} v_j,$$

i.e., we have $v_i = l$ for all $i \notin M$ and some l.

By (i) and (ii), we have

$$\sum_{j\notin M} v_j = (n-m)l \text{ and } \sum_{j\notin M} v_j = \lambda k,$$

$$\sum_{j=1}^{n} v_j = \sum_{j\in M} v_j + \sum_{j\notin M} v_j = mk + (n-m)l \text{ and } \sum_{j=1}^{n} v_j = \lambda l,$$

i.e.,

$$(n-m)l = \lambda k, \tag{11}$$

$$mk + (n-m)l = \lambda l. \tag{12}$$

With $\lambda \neq 0$ and $v \neq 0$, it follows immediately from (11) that $k \neq 0$ and $l \neq 0$. Now, combining (11) and (12) yields

$$m\frac{(n-m)l}{\lambda} + (n-m)l - \lambda l = 0$$

$$\overset{l\neq 0}{\Leftrightarrow} m(n-m) + (n-m)\lambda - \lambda^2 = 0$$

$$\Leftrightarrow \lambda_{1,2} = \frac{1}{2}((n-m) \pm \sqrt{(n+3m)(n-m)})$$

Since $m < n$, we have $\lambda_1 \neq \lambda_2$ and $\lambda_1, \lambda_2 \neq 0$. Since $k \neq 0$, and any non zero scalar multiple of an eigenvector v is also an eigenvector, we may assume that $k = 1$. Thus, we have (cf. (11)) $l = \lambda/(n-m)$ for $\lambda \in \{\lambda_1, \lambda_2\}$. We obtain eigenvectors

$$v_i^1 = \begin{cases} 1 & \text{if } x_i = 1, \\ \dfrac{\lambda_1}{n-m} & \text{if } x_i = 0, \end{cases}$$

$$v_i^2 = \begin{cases} 1 & \text{if } \quad x_i = 1, \\ \dfrac{\lambda_2}{n-m} & \text{if } x_i = 0, \end{cases}$$

$(i = 1, \ldots, n)$ associated with λ_1 and λ_2, respectively. It is easy to check that the vectors v^1 and v^2 are orthogonal, i.e., we have $\sum_{i=1}^{n} v_i^1 v_i^2 = 0$. Moreover, we have $\|v^1\| = \sqrt{2m+\lambda_1}$ and $\|v^2\| = \sqrt{2m+\lambda_2}$. Thus, normalizing v^1 and v^2, we obtain orthonormalized eigenvectors

$$\begin{aligned} v^1 = (v_i^1) &= \frac{s_1}{\sqrt{2m+\lambda_1}}\left(\frac{\lambda_1}{n-m} + \left(1 - \frac{\lambda_1}{n-m}\right)x_i\right) \\ &= \frac{s_1}{(n-m)\sqrt{2m+\lambda_1}}(\lambda_1 + \lambda_2 x_i) \\ &= s_1 \frac{\lambda_1 + \lambda_2 x_i}{(n-m)\sqrt{2m+\lambda_1}}, \end{aligned}$$

$$v^2 = (v_i^2) = \frac{s_2}{\sqrt{2m+\lambda_2}}\left(\frac{\lambda_2}{n-m} + \left(1 - \frac{\lambda_2}{n-m}\right)x_i\right)$$

$$= \frac{s_2}{(n-m)\sqrt{2m+\lambda_2}}(\lambda_2 + \lambda_1 x_i)$$
$$= s_2 \frac{\lambda_2 + \lambda_1 x_i}{(n-m)\sqrt{2m+\lambda_2}}$$

for $i = 1, \ldots, n$ and $s_1, s_2 \in \{-1, 1\}$ associated with λ_1 and λ_2, respectively. □

References

[1] E.R. Barnes, An algorithm for partitioning the nodes of a graph, SIAM J. Algebra Discrete Math. 3 (1982) 541–550.

[2] E.R. Barnes and A.J. Hoffman, Partitioning, spectra, and linear programming, in: R.W. Pulleyblank, ed., Progress in Combinatorial Optimization (Academic Press, Canada, 1984) 13–25.

[3] M. Blum, R. Floyd, V. Pratt, R. Rivest and R. Tarjan, Time bound on selection, J. Comput. and System Sci. 7 (1972) 240–267.

[4] P. De, J.B. Gosh and C.E. Wells, CON due-date determination and sequencing, Comput. Oper. Res. 17 (1990) 333–342.

[5] P. De, J.B. Gosh and C.E. Wells, On the minimization of completion time variance with bi-criteria extension, Oper. Res. 40 (1992) 1148–1155.

[6] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, 1979).

[7] F. Glover, E. Taillard and D. De Werra, A user's guide to Tabu search, Ann. Oper. Res. 41 (1993) 3–28.

[8] N.G. Hall and M.E. Posner, Earliness-tardiness scheduling problems, I: weighted deviation of completion times about a common due date, Oper. Res. 39 (1991) 836–846.

[9] P. Hansen, B. Jaumard and V. Mathon, Constrained nonlinear 0–1 programming, ORSA J. Comput. 5 (1993) 97–119.

[10] R.A. Horn and C.A. Johnson, Matrix Analysis (Cambridge Univ. Press, Cambridge, 1985).

[11] J. Józefowska and W. Kubiak, Spectral algorithms for the completion time variance problem, Working Paper, Faculty of Business Administration, Memorial University of Newfoundland (1993).

[12] H.G. Kahlbacher, SWEAT - a program for scheduling with earliness and tardiness penalties, European J. Oper. Res. 43 (1989) 111–112.

[13] J.J Kanet, Minimizing variation of flow time in single machine systems, Management Sci. 27 (1981) 1453–1459.

[14] N. Karmakar and R.M. Karp, The differencing method of set partitioning, Report UCB/CSD 82/113, Computer Science Division, Univ. of California, Berkeley (1982)

[15] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., Complexity of Computer Computations (Plenum Press, New York, 1972) 85–103.

[16] W. Kubiak, New results on the completion time variance minimization, Discrete Applied Math. 58 (1995) 157–168.

[17] W. Kubiak, Completion time variance minimization on a single machine is difficult, Oper. Res. Lett. 14 (1993) 49–59.

[18] E.L. Lawler and J.M. Moore, A functional equation and its application to resource allocation and machine scheduling problems, Management Sci. 16 (1969) 77–84.

[19] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, Ann. Discrete Math. 1 (1977) 343–362.

[20] S. Martello and P. Toth, Knapsack Problems (John Wiley, Chichester, 1990).

[21] B. Mohar and S. Poljak, Eigenvalues in combinatorial optimization, preprint (1992).

[22] F. Rendl and H. Wolkowicz, A projection technique for partitioning the nodes of a graph, Research Report, University of Waterloo (1990).

[23] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logist. Quart. 18 (1956) 295–305.